Prototyping IoT with yocto PROJECT

Pierre Ficheux (pierre.ficheux@smile.fr)

02/2017

$ whoami

- French embedded Linux developer, writer and teacher
- CTO @ Smile-ECS (Embedded & Connected Systems)

2 kinds of objects

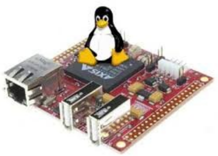
- Basic one such as sensor
  - MCU/µC (no MMU)
  - Software is « bare metal » or light OS such as Contiki or RIOT
- Advanced one (computer like)
  - CPU with MMU (32 bits or more)
  - OS such as Linux / Tizen / Android

Parrot flower power (µC)

“Tesla car is a connected computer on wheels !”

Eccelenza touch (Yocto)

Linux and IoT

- Not “the” universal OS for IoT but...
- According to “IoT developer Survey 2016”
  - 73 % Linux
  - 23 % « bare metal » (no OS)
  - 12 % FreeRTOS
  - 6 % Contiki
- Don't forget there are  and 
  - Distribution (Debian, Ubuntu, etc.)
  - « Build system » (Yocto, Buildroot, etc.)
- Today most of objects are computers

Linux distribution

- Most of developers use Linux distribution
- Well known, comfortable and portable environment but
  - High footprint (Go)
  - boot time (close to 1 mn)
  - Development oriented → host but not a target
  - No traceability (binaries)
  - Limited target support (x86, ARM)
  - Not for IoT at all !!
- Most distributions runs on ARM → easy to take a wrong way
- Alternate – and right - way is « build system » !

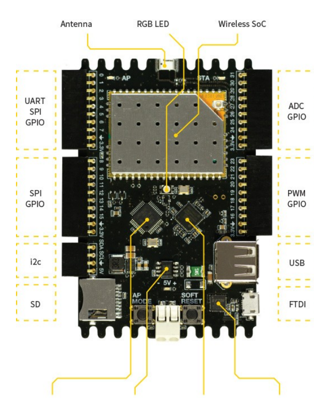
What is a « build system » ?

- Not a distribution, just a tool to build one from sources
- Does not provide sources but “recipes”
- Provides binaries file to be installed on the target
  - Bootloader
  - Linux kernel and DT blobs
  - Root-filesystem image + applications
- Provides additional information
  - Licensing
  - Dependencies graphs
- Much better footprint, boot time, etc.
- Android uses a dedicated – but open source - build systems

Smile ECS Embedded & Connected Systems

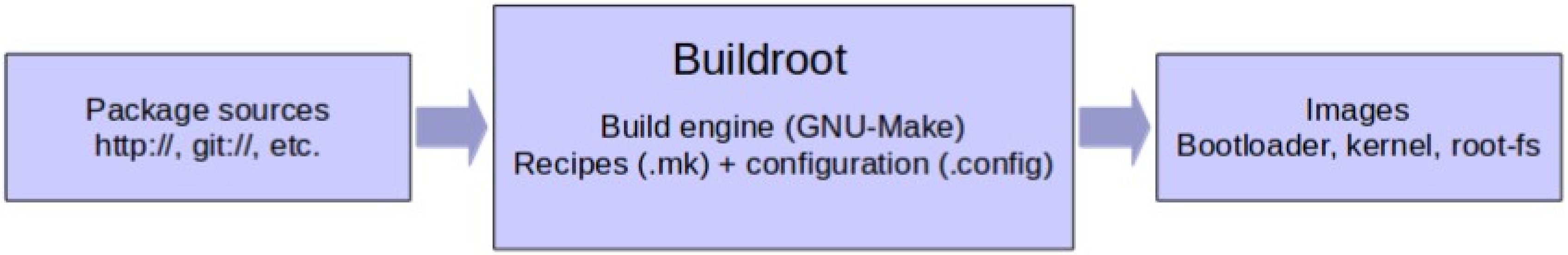
Most famous build systems

- Yocto/OpenEmbedded
  - Based on “BitBake” (Python)
  - Very powerful, not that easy to learn
  - Text oriented
- Buildroot
  - Based on standard GNU Make
  - Started as an internal tool for uClibc
  - Static approach (no packages)
- OpenWrt
  - Modified Buildroot
  - Packaging support
  - Used for WeIO (IoT device)



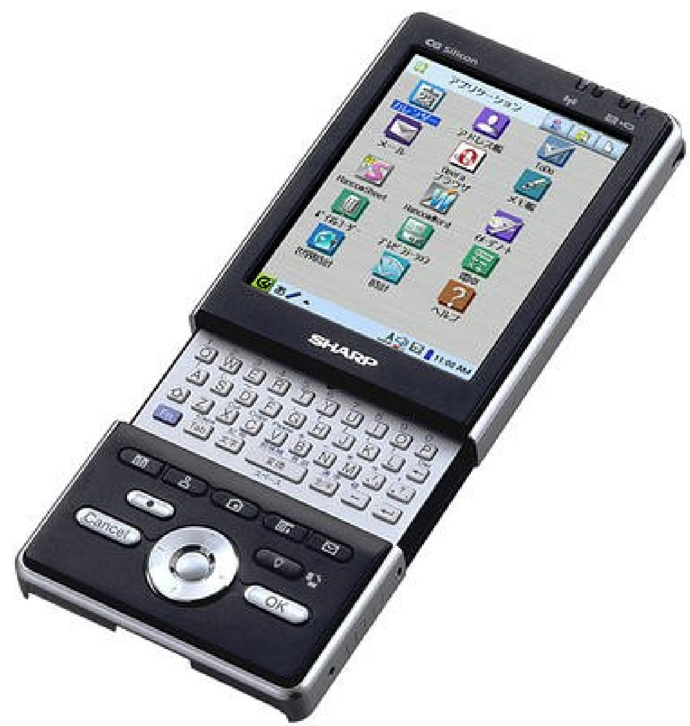
Buildroot

- Formerly internal tool for uClibc
- One version every 3 months since 2009.02
- Kernel like graphical configurator
- Fast and easy to use
- Result is not a distribution but a “Linux firmware”



OpenEmbedded

- A “cross compilation framework”
- Started Chris Larson, Michael Lauer et Holger Schuring for “OpenZaurus” (2002)
- Zaurus (SHARP) was the “first” Linux/Qt PDA

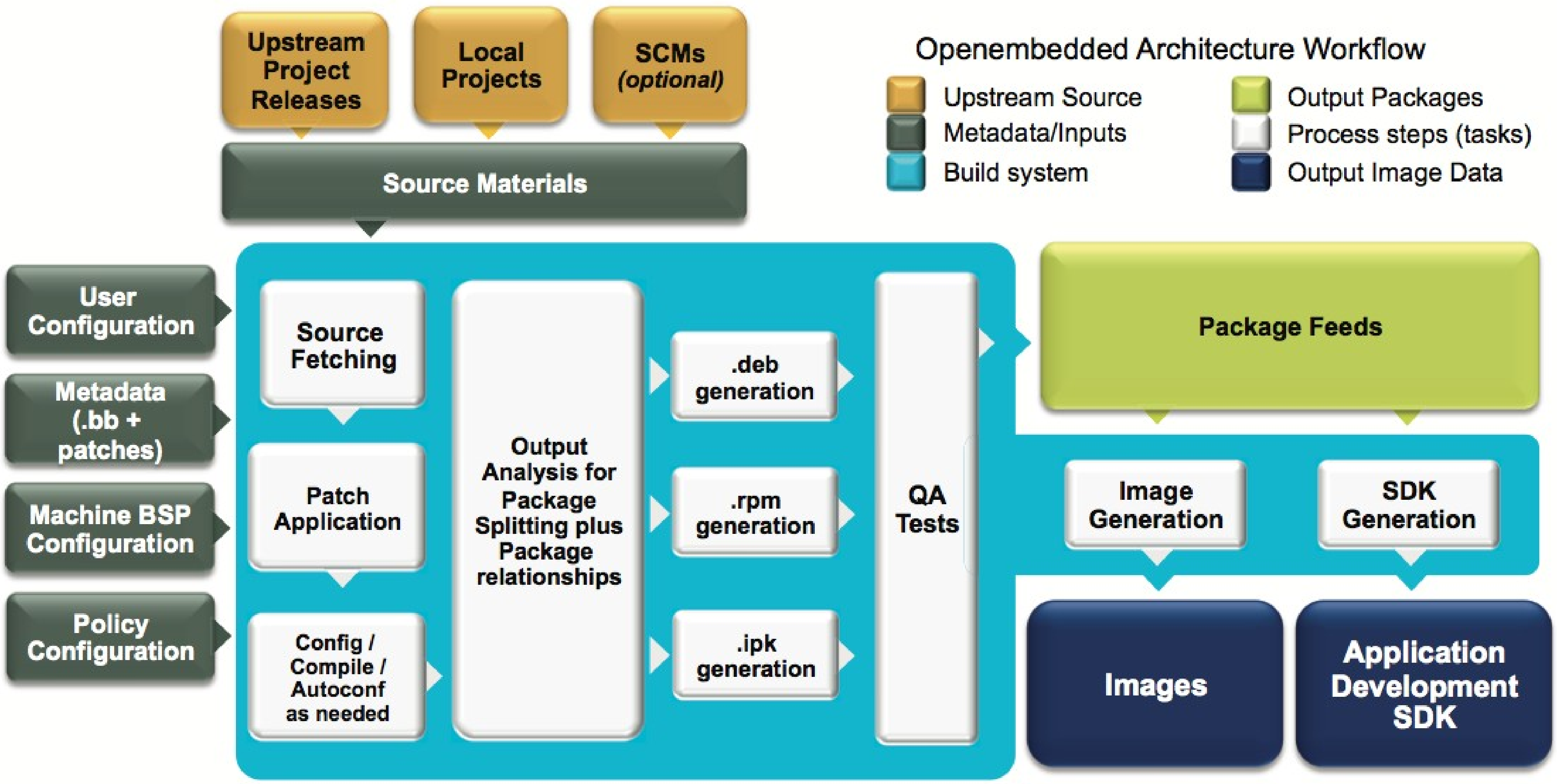


OE principles

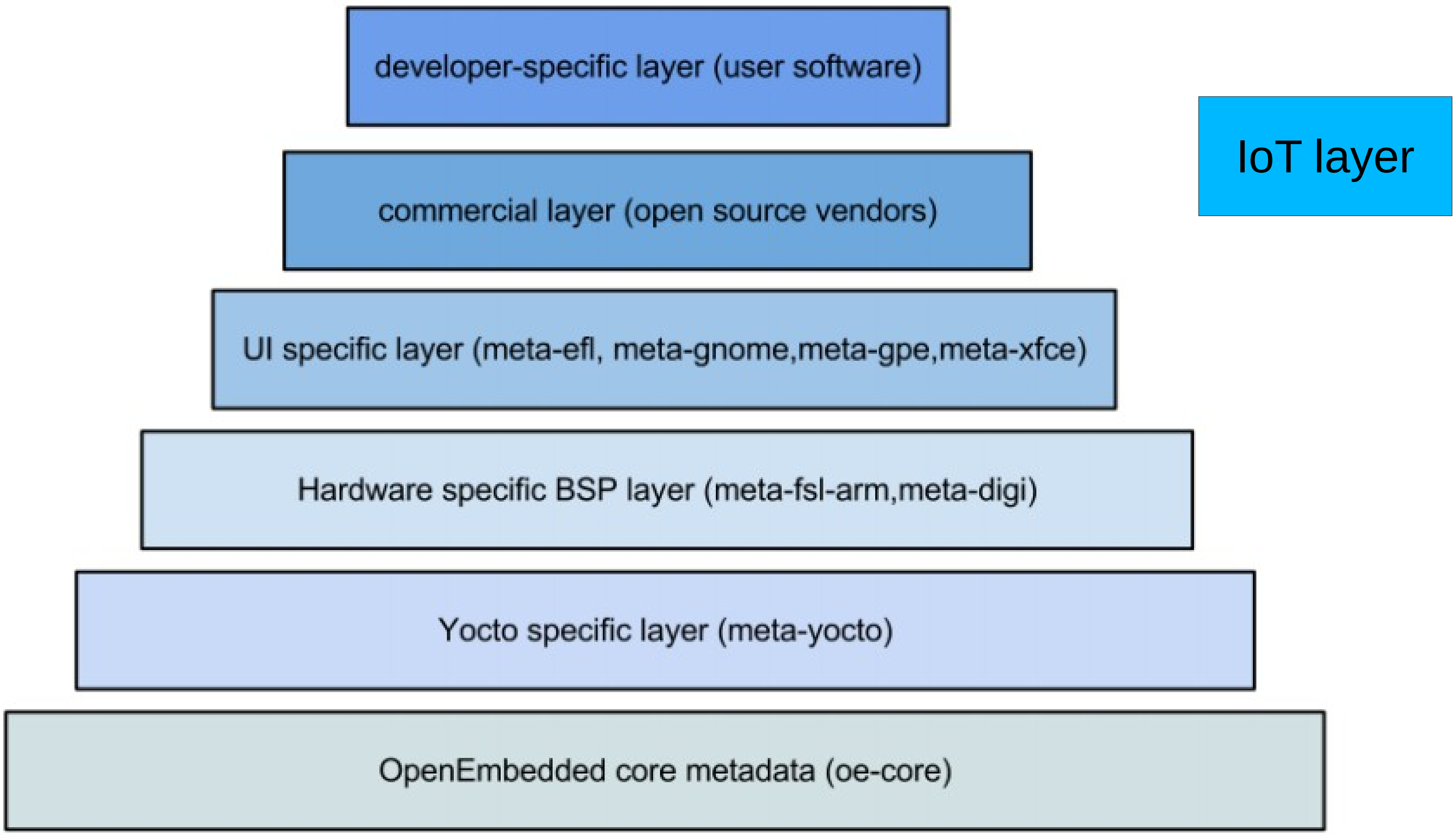
- Recipe is a `.bb` (for BitBake) file for every component (from “Hello World” to whole distribution)
- OE uses classes (`.bbclass`), headers (`.inc`) and configuration files (`.conf`)
- You can inherit from class with `inherit`
- “Deriving” a recipe is VERY useful → `.bbappend`
- Files are organized as “layers” → `meta-*`
- OE data flow is based on packages (RPM, IPK, DEB)
- Package management on target is optional

- Yocto (symbol y) is a unit prefix in the metric system denoting a factor of 10^{-24}
- Yocto project was started in 2010 by Linux foundation
- Sub-projects integration (OE, BitBake, Poky, etc.)
- Currently most of embedded companies and hardware makers are members (Intel, Montavista, NXP, TI, etc.)
- Richard Purdie (Linux Foundation fellow) is the architect
- Most of Linux BSP are provided as OE layers !

Yocto / OE workflow



Yocto / OE layers



Yocto/Poky « in a nutshell »

- Installing Poky and BSP

```
$ git clone -b krogoth git://git.yoctoproject.org/poky
$ cd poky
$ git clone git://git.yoctoproject.org/meta-raspberrypi
```

- Creating working directory

```
$ source oe-init-build-env rpi-build
```

- Adding BSP layer to `conf/bblayers.conf`

```
$ bitbake-layers add-layer meta-raspberrypi
```

- Adding target name to `conf/local.conf`

```
MACHINE = "raspberrypi"
```

- Creating minimal image

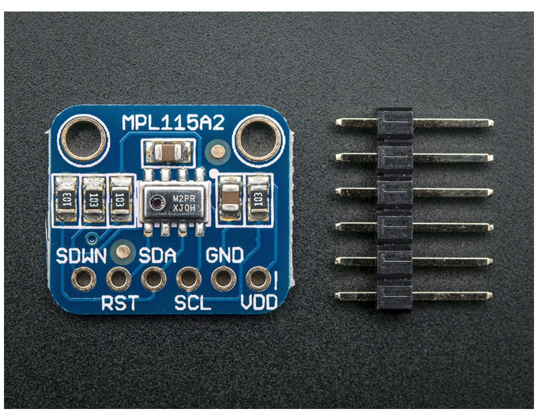
```
$ bitbake core-image-minimal
```

- Testing on SD card

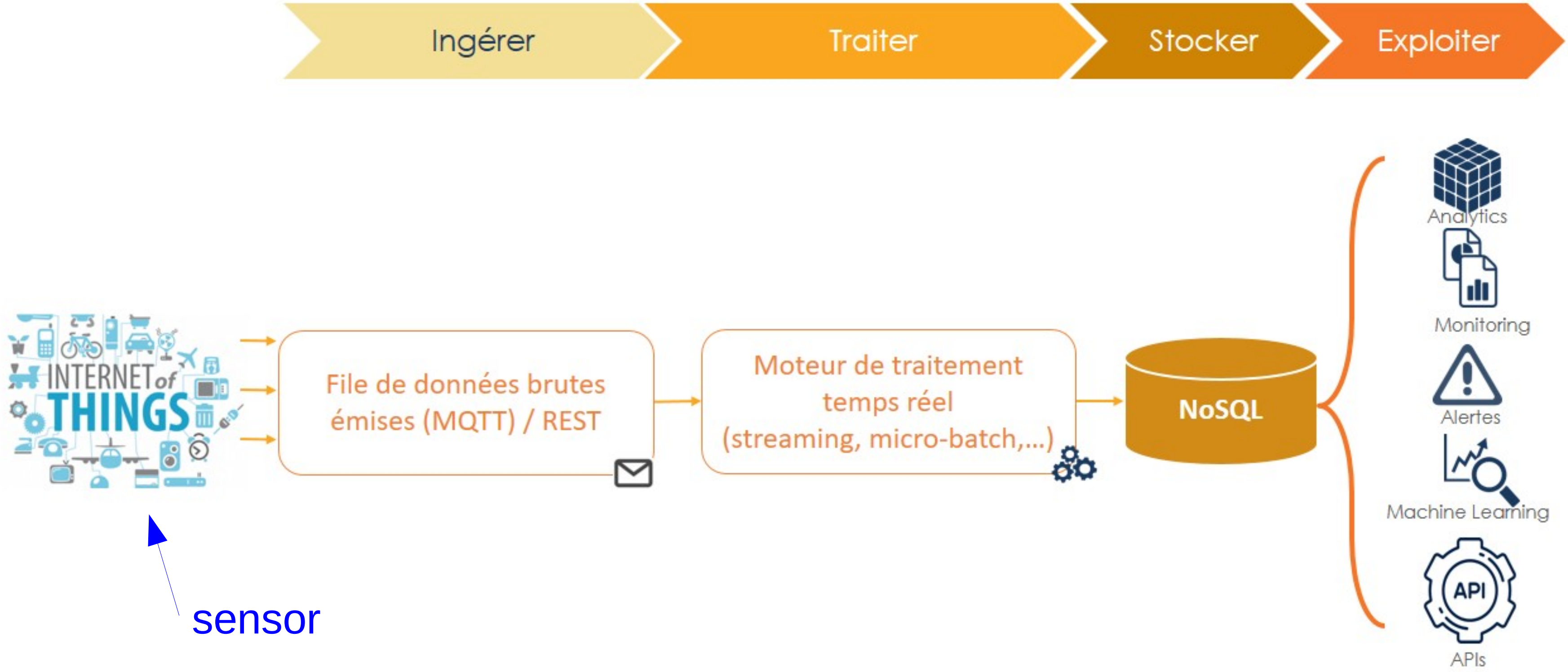
```
$ sudo dd if=<path>/core-image-minimal-raspberrypi.rpi-sdimg
of=/dev/sdb
```

Use case 1 : IoT sensor

- Building a demo sensor for Smile
  - Raspberry Pi (zero)
  - I²C temperature/pressure sensor (MPL115A2)
  - Wi-Fi (USB)
  - HTTP protocol



Demonstrator global architecture



Building distribution

- Starting from smaller distro « core-image-minimal »
- Adding options and new recipes
  - Package management
  - Standard or “derivated” recipes
  - New recipes (I²C sensor control)
- Put everything in a new layer → `meta-iot`

```
$ yocto-layer create iot
```

- Updating `local.conf` (for test only)
- Creating a new distro recipe → « rpi-iot-image »

Using `.bbappend` (derivation)

- One recipe (`.bb`) is defined in layer “A”
- We update recipe in a `.bbappend` located in layer “B”
- Currently
  - Network configuration (Wi-Fi + HTTPd)
  - I²C activation in `config.txt`
  - Autoload of *i2c-dev* module

Wi-Fi + HTTPd

- Wi-Fi adapter is supported → `wlan0`
- We need some additional packages (Wi-Fi management + HTTP server=

```
IMAGE_INSTALL_append += "iw wpa_supplicant lighttpd"
```

- Updating `/etc/network/interfaces` for `wlan0` automatic configuration
- WPA authentication (manual procedure for test)

```
# wpa_passphrase <ESSID> <password> > /etc/wpa_supplicant.conf
# ifdown wlan0
# ifup wlan0
```

Sensor + I²C

- Updating `config.txt`

  `dtparam=i2c_arm=on`

  → `do_deploy_append()`
- Adding packages to `local.conf`

  `IMAGE_INSTALL_append += "i2c-tools kernel-modules"`
- Loading I²C support

  `KERNEL_MODULE_AUTOLOAD += "i2c-dev"`

  → Kernel `.bbappend`
- New recipe for MPL115A2 control
  - Adapting original program (C, based on WiringPi)
  - Starting a “service”, reading sensor every 20 secs
    → using *update-rc.d* class

NTP support

- No RTC on Raspberry Pi
- NTP recipe provided by *meta-openembedded* layer

```
$ cd poky
$ git clone git://git.openembedded.org/meta-openembedded
$ git checkout <yocto-branch>
$ bitbake-layers  add-layer ../meta-openembedded/meta-oe
$ bitbake-layers  add-layer ../meta-openembedded/meta-python
$ bitbake-layers  add-layer ../meta-openembedded/meta-networking
$ bitbake ntp tzdata
```

- Configuring timezone

```
# rm -f /etc/localtime
# ln -s /usr/share/zoneinfo/Europe/Paris /etc/localtime
# cat /etc/default/ntpdate
...
NTPSERVERS="pool.ntp.org"
```

Updading target

- SMART included by package management
- Creating packages index

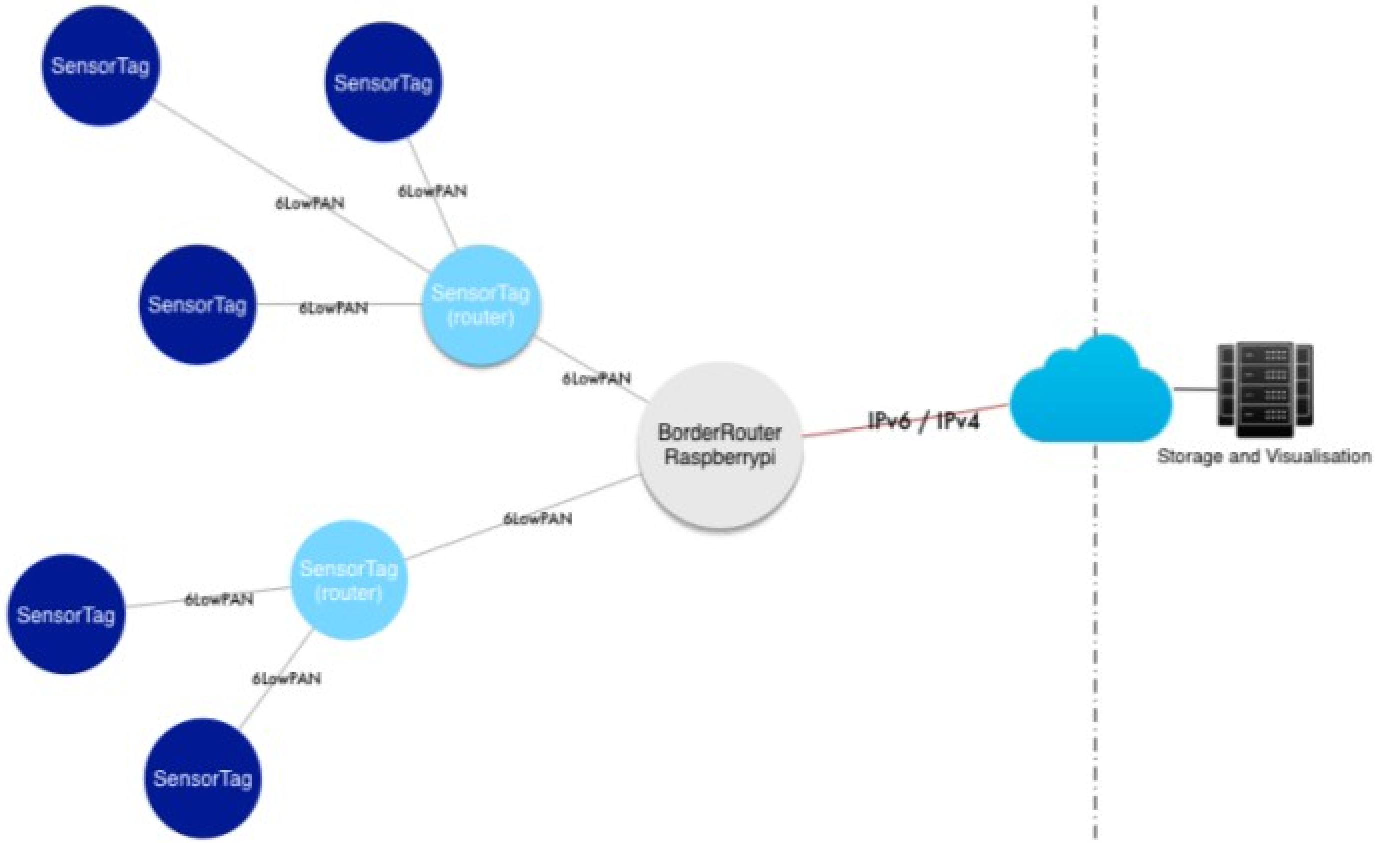
```
$ bitbake package-index
```

- Creating HTTP channels on the target

```
# smart channel --add <channel> baseurl=http://<pkg-dir>
# smart update
# smart install ntpdate tzdata
```

Use case 2 : Border router (N. Aguirre)

- More complex demonstration based on sensorTag (TI)
- Raspberry Pi (Yocto 2.1 based) as “border router”

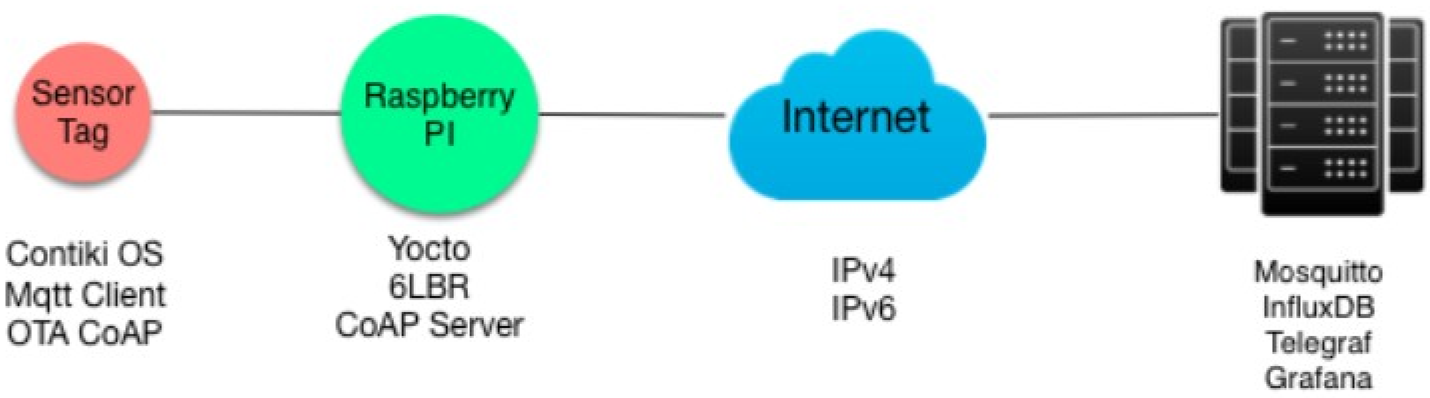


SensorTag

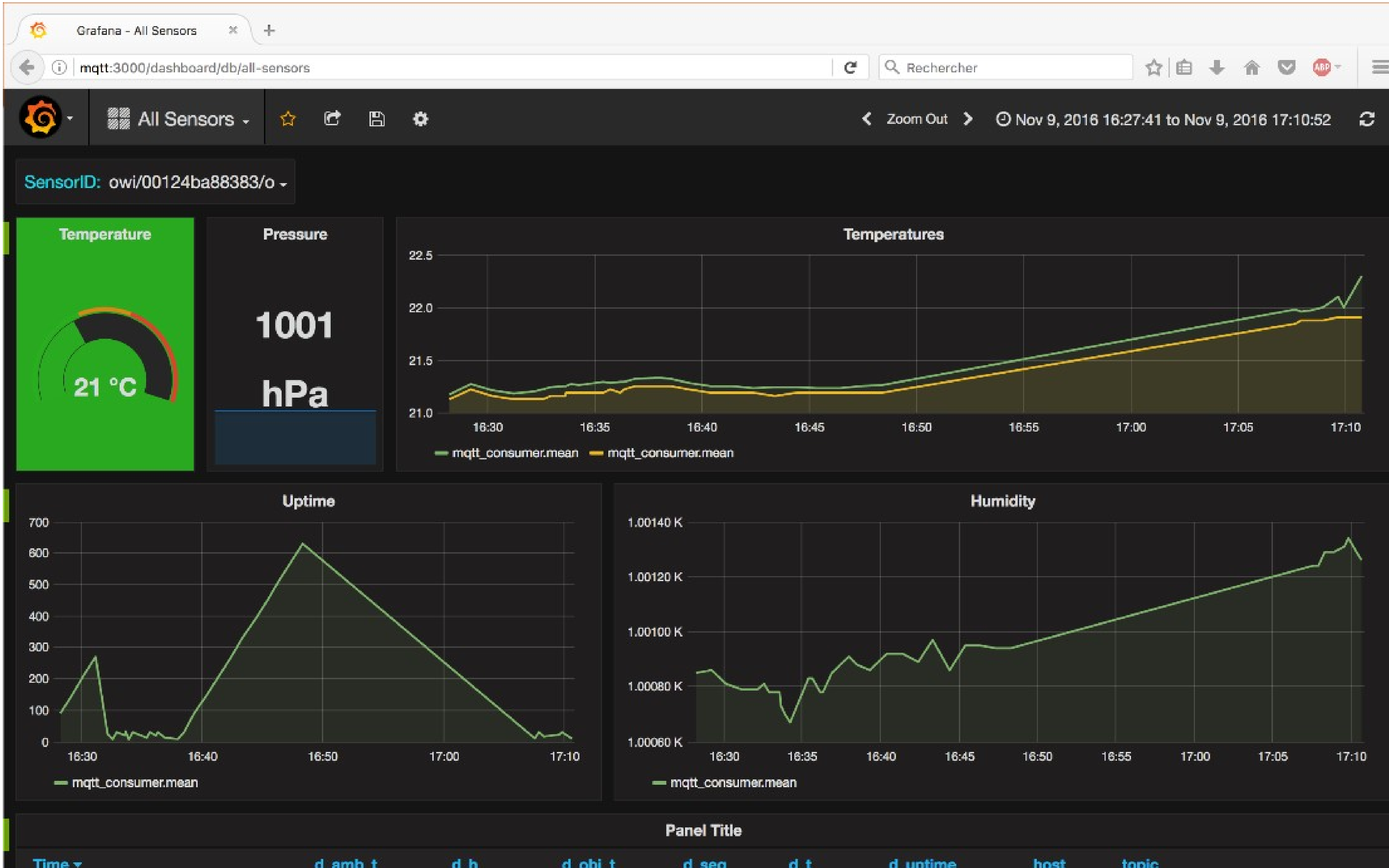
- Cortex M3 (48MHz, 128KB flash, 8KB RAM)
- 512KB external flash for OTA and/or storage
- Low-power (10 mA active, 100 uA sleeping)
- Radio 802.15.4 + Bluetooth Low Energy (BLE)
- $ 30 from TI website

Raspberry Pi + 6LBR

- 6LBR est a board router software (between IoT/sensors world and Internet world)
- Get data from SensorTags (6LoWPAN)
- Send data to the “cloud”
- MQTT broker
- Time Series (Influxdb) database
- MQTT / database connector (Telegraf)
- Web management and display (Grafana)



Grafana display



Références

- http://elinux.org/Build_Systems
- https://www.yoctoproject.org/
- http://buildroot.uclibc.org
- http://iot.ieee.org/images/files/pdf/iot-developer-survey-2016-report-final.pdf
- https://openwrt.org
- http://eccellenzatouchvki.com
- http://www.parrot.com/fr/produits/flower-power
- https://www.yoctoproject.org/ecosystem/iot
- http://we-io.net/hardware
- https://github.com/nodesign/weioBoard
- https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/README.md
- http://www.ti.com/ww/en/wireless_connectivity/sensortag2015